Group Decision and Negotiation (2007) 16:169–190
DOI 10.1007/s10726-006-9061-7

The Challenge of Self-adaptive Systems for E-commerce

HANS WEIGAND
INFOLAB, Tilburg University, PO Box 90153, Tilburg, 5000 LE, The Netherlands (E-mail: weigand@uvt.nl)

WILLEM-JAN VAN DEN HEUVEL
INFOLAB, Tilburg University, PO Box 90153, Tilburg, 5000 LE, The Netherlands (E-mail: wjheuvel@uvt.nl)

Abstract

For future E-commerce systems that are engaged in many dynamic trading relationships, the ability to adapt themselves smoothly will increasingly become a critical property. In this paper, we first define the basic semantic structure of a collaborative process. Then we introduce a formal framework for self-adaptive systems. We argue that self-adaptive systems should specify goals explicitly, and propose a goal-based architecture. We further argue that for systems that operate in a shared environment with other systems, self-adaptation should be extended with co-adaptation. We define four levels of co-adaptation, and present an argumentation mechanism that can be used to enable co-adaptation at the higher levels.

Key words self-adaptive systems, co-adaptation

1. Introduction

Enterprises are teaming up to better cope with the ever-more increasing customer requirements and become more competitive, defining and executing collaborative processes with potentially unknown trading partners on the fly (Smith and Fingar 2003). Unfortunately, these collaborative processes, also called c-processes, are not stable, but tend to be in flux all of the time, e.g., to conform to evolving industry standards, accommodate new partners, or to be aligned with changed or new legal rules. This implies that collaborations need to be adapted continually. Given the current portfolio of enterprise applications, that is characterized by fragmentation of "frozen" business processes over multiple (stovepipe) applications, redundancy of applications and database systems, and many complex interfaces to enhance inter-operation, this is increasingly becoming a Herculean, but nevertheless business-critical, task. Drawing on a recent study by IBM, Salehie and Tahvildari state that 40% of all investments in IT are used just trying to get technologies to work together (Salehie and Tahvildari 2005).

According to current estimates, the maintenance of systems already makes up about 70–80% of all costs during an enterprise application's lifecycle (Pfleeger and Atlee 2006). In order to cut down costs of rather tedious and labor-intensive routine

maintenance tasks, and establish more effective mechanisms to deal with change, *autonomic computing* is touted in industry as an effective solution (Kephart and Chess 2003). In a nutshell, autonomic computing provides a contemporary paradigm for managing computing resources, eliminating the need for human interference (Ganek and Corbi 2003). This is to be achieved by giving enterprise systems not only full awareness about their internal model, but, also the ability to adapt themselves, particularly on non-functional properties such as performance, fault-tolerance and security. However, it is fair to say that autonomic computing is mainly a vision, and there is not much of an actual solution to be shown yet.

The main objective of this paper is to develop and explore a framework for self-adaptive systems to leverage interoperability with other systems. It also introduces some solution components. In particular, we will argue that in the context of collaborative processes, self-adaptation cannot be implemented effectively without some level of co-adaptation, which in turn requires a generic argumentation system to be in place. The practical relevance of this paper is that it provides architectural patterns for self-adaptation and some solution ingredients. The theoretical relevance is that it clarifies the concept of self-adaptation and what is needed to achieve self-adaptation, particularly in an environment of mutually dependent interacting systems.

The structure of this paper is as follows. In Section 2, we introduce a basic logical framework for collaborative processes. In the next three sections, we build up a self-adaptation framework in three steps: first, in Section 3, we explore adaptability, next, in Section 4, we present a framework for self-adaptation, and we extend this with a framework for co-adaptation in Section 5. In Section 6 work out one solution component for co-adaptive systems in the form of an argumentation system. Section 7 concludes with a summary of results and an overview of future research.

2. A Logical Framework for Collaborative Processes

In this section, we present a basic semantic framework for collaborative processes that provides the context in which we want to explore self-adaptation and co-adaptation.

In Weigand, Verharen and Dignum (1997), a formal language called L_{ill} is described with which an integrated semantics for information and communication systems can be expressed. It is an extension of Dynamic Deontic Logic and the semantics of speech acts is described using preconditions and postconditions. For example, the postcondition of an authorized request is that the Hearer is obliged to perform the requested action. Pre- and post-conditions have been used also in agent communication languages such as KQML and FIPA-ACL (Chaibdra and Dignum 2002). For example, the precondition of KQML's *tell* message states that the sender believes what it tells and that it knows that the receiver wants to know that the sender believes it. The postcondition of sending the *tell* message is that the receiver can conclude that the sender believes the content of the message. In a similar vein, FIPA-ACL uses feasibility preconditions and rational effects. There

have been critical discussions about this approach (Chaibdraa and Dignum 2002). Some have argued that the semantics should not be based on mental states, but on social commitments (Singh 2000). Others have proposed to ground the semantics in the notion of sign conventions (Jones and Parent 2003). The semantics that we propose here is in line with these latter two approaches in the sense that we agree that the effect on the social world should be at the core. This is also in accordance with Habermas's theory of communicative action (Habermas 1984). Where our approach differs from the latter two is the Habermasian assumption that intersubjective truth (common ground) is established by a joint act of speaker and hearer (cf. Clark 1996).

The semantics that we propose in this paper is built on this assumption. The theory of communicative action is based on the notion of validity claim. Speakers make claims, and when these claims are accepted or conceded, they turn into common ground. In this way, coordination can be achieved. The general scheme is as follows.

Definition
Inference scheme for communication semantics.
For all .φ being a well-formed formula, i,j being conversational roles
[claim(i,j, φ); accept(j,i, φ)] $\text{agreed}_{i,j}(\varphi)$
end

In words: when φ has been claimed and has been accepted, then it is agreed. For example, when some customer claims that the seller should take care of the insurance of the transport, and the seller accepts this claim, (only) then there is an agreed upon obligation. Evidently, there must be a ground for the customer claim, so in an abstract sense, as a default, the obligation can be said to exist before the agreement, but there may always be specific circumstances that invalidate the default obligation. The effect of the communicative action is that the obligation is materialized.

The modality *agreed* is rather weak from a logical perspective. It does adhere to conjunction distribution

$$\text{agreed}(\varphi \wedge \psi) \leftarrow \text{agreed}(\varphi) \wedge \text{agreed}(\psi)$$

but (just as knowledge and belief modalities in modern epistemic logic) it is not necessarily closed under implication:

$$\text{agreed}(\varphi \rightarrow \psi) \wedge \text{agreed}(\varphi) \rightarrow \text{agreed}(\psi)$$

In terms of Habermas, the *agreed* is also described as the situation definition. In this paper, we will use the term *shared model* for what is agreed upon at the instance level, and *common ground* for the agreed upon rules and specifications (Stamper 2000).

The fact that φ has the status "agreed" does not say anything about internal beliefs. Depending on whether the hearer is convinced of the sincerity and trustworthiness of the speaker, he will infer (or not) that φ is believed by the speaker and believe it himself. This inference may be important, but it is not critical for the

conversation process itself, because what counts for the communication partners is what is agreed upon.

Using the basic communication semantics above, we are able to describe the effects of conversational acts once we know which claims are made with a certain conversation act. Among the many claims that could be made by a speaker when performing a conversational act, we distinguish the following essential categories:

- *claims about the "agreed"* – such as presuppositions. The object of the claim can be many kind, including the categories given below. To distinguish these claims from the new claims, we use the modality *agreed.*
- *references* – when performing conversational acts, the speaker has to refer to certain objects: a person, a product item, a delivery etc. By referring to an object, the speaker claims that this object exists, and by accepting this claim, the object becomes part of the shared model. The next time the speaker refers to the same object, this reference is an agreed reference and the claim falls in the first category. Under "references" we also categorize identity claims about objects of the form x = y.
- *authorization claims* – when performing a conversational act, the speaker claims that he is authorized to perform the act.
- *claims about actor obligations* – what a actor or set of actors should do. Obligations can be in one of the following states: *created, cancelled, violated, fulfilled.*
- claims about actions to be performed – the things to be achieved in the business process: *desired, intended, started, finished, approved.* These phases correspond roughly to the well-known action cycle of Norman (Norman 1990).

This categorization is not meant to be exhaustive, but it covers the most important cases in our context. For each of the *claim* types, there is also a corresponding *accept* action.

Example

We consider a purchase event. When the Buyer X.com sends the purchase order to the Seller Y.com, this is a request for a delivery *d* containing the following claims:

- references:
  - ◯ *object*(*Buyer, subject*), *object* (*Seller, subject*),
  - ◯ *object*(*X.com, subject*), *object*(*Y.com, subject*),
  - ◯ *object*(*d, delivery*),
  - ◯ *Buyer = X.com, Seller = Y.com*

- *authorized(Buyer,PurchaseOrder)* Buyer is permitted to make this request.
- *created(obligation(Seller,d))* Seller is obliged to bring about *d*, that is, to deliver the requested goods.
- *intended(d)* the action state of the event d is "intended" ("to be done")

The conversation starts with the action status of *d* being already "desired". This may have been established earlier in the conversation; otherwise this claim functions as a presupposition. The Buyer claims that the delivery is to be performed now (intended), and claims that the Seller (with executor role) is obliged to perform it. Both claims can be challenged by the Seller, but if they are accepted, they lead to an obligation for the executor and a state change of the action itself (from intended to started).

The obligation claims and action state claims are closely related, because it would be odd when an action is considered to be intended, but no one is responsible for the execution, or vice versa. However, these odd situations can happen in complex settings. For example, when the Seller withdraws after having committed because of force major and his obligation is cancelled. By separating the two claims, it also becomes possible to accommodate the situation that the hearer accepts one claim but challenges the other.

3. A Framework for Adaptability

A business collaboration is typically defined in the form of a protocol, message and data type definitions, domain ontologies, and collaboration profiles (ebXML) or a contract. Collectively these definitions establish a shared syntax for the message exchange and a shared semantics. The definitions can cover functional and non-functional aspects and establish the common ground for the interaction to be successful. In current E-commerce systems, the definitions are often specified in some proprietary XML-based language that is relatively close to the actual implementation and notably hard to reconfigure and change, but there is a tendency to conceptualize definitions using formal languages that allows logical inferences, and do not assume in-depth knowledge about implementation-level communication and transaction protocols that are used, nor the component model that is chosen (e.g., .NET or J2EE). Examples of higher-level specifications of (parts of) business collaborations include FMEC (Kimbrough and Moore 1997), the layered approach of (Weigand and Van den Heuvel 1999) and the contract language BCL (Linington et al., 2004). The advantage of such a high-level specification is similar to the advantage of "architecture model-based" approach advocated in the self-adaptive system literature (Garlan and Schmerl 2002; Valetto and Kaiser 2002) and the MDA approach (Kleppe, Warmer and Bast 2003) in the field of interoperable systems.

Even when such languages are used however, adaptation of definitions remains a delicate process. A fundamental problem, in our view, is that of *effect indeterminacy*. Given a desired change, it is not easy to find out which part of the specification should be adapted, and given a certain change in the specification, it is not easy to know the effect that the change has on the rest of the specification and the implementation. We do not pretend to have an overall solution for the effect indeterminacy problem (which can be seen as a variant of the wicked "frame problem" in AI), but suggest that a *goal-based* architecture provides part of a solution to this problem, as such architecture tells you the purpose of a certain component and what are possible alternatives. It means that the specification is organized using a goal/means

structure that has been explored extensively in so-called goal-oriented requirements engineering (cf. Lamsweerde 2001; Mylopoulos, Chung and Yu 1999). One may distinguish between hard goals and soft goals. A hard goal includes a clear specification about when it is actually reached, while a soft goal does not have such clear criteria. Some authors distinguish between operational and strategic goals, formulating a strategic goal as a non-operational objective and operational goals as constraints on the actions that can be performed by a system (Dardenne, van Lamsweerde and Fickas 1993). Goals are attained using some means involving plans, recourses and soft goals. Often, one goal may be achieved by applying alternative means, and goals may be recursively decomposed into sub-goals (refinement), resulting into a goal hierarchy with OR and AND branching.

In the context of c-processes, it is useful to make an explicit distinction between collaborative goals, also named interaction goals (Cheong and Winikoff 2005), and system goals. Collaborative goals are shared between two or more (trading) partners expressing what they would like to attain by collaborating, while system goals denote goals of individual actors. In order to effectuate c-processes, system goals should be aligned with collaborative goals.

To set up a goal hierarchy for a certain application, such as a c-process, will be quite an investment, as this knowledge is often left implicit. However, given such a structure it becomes feasible to replace one ''means'' by another ''means'' if the new means satisfies at least the same goals as the old means. The required satisfaction must be ascertained or validated somehow.

There are three ways to achieve this validation: by formal constraint checking, by certification, or by user approval. The first approach requires that the satisfaction criteria be formalized in a logical language that allows automated model checking. For certain properties, this is an attractive option, but based on past experience, we think it will provide only a partial solution. The second approach requires that trustworthy statements can be acquired, from some certification authority, about what the component supports (e.g. this component supports that protocol, or this protocol supports non-repudiation as defined in standard X). This should be explored as much as possible, but we can expect it to work for generic components only. The third option is to rely on the user. In a self-adaptive system, we should try to minimize the user involvement. So, given the limitations of each approach, we need a careful combination of all three.

We propose a goal hierarchy based on two dimensions, one for the functional goals and one for the non-functional goals. From a language/action perspective (Dietz 2005), at least three levels can be distinguished in the functional goal structure: the *collaboration* level, the *symbolic* level and the *physical* level (see Figure 1). At the collaboration level, the business transaction is described in terms of value exchanges and communicative actions; these business transactions are realized by information exchange, and this is the second symbolic level. Finally, the information must somehow be transferred physically as messages via a computer network, or by paper. This is addressed at the physical level. We have added a fourth *context* layer on top of the collaboration level that describes the values and cultural norms of the society in which the collaboration takes place, as well as the strategic goals of each

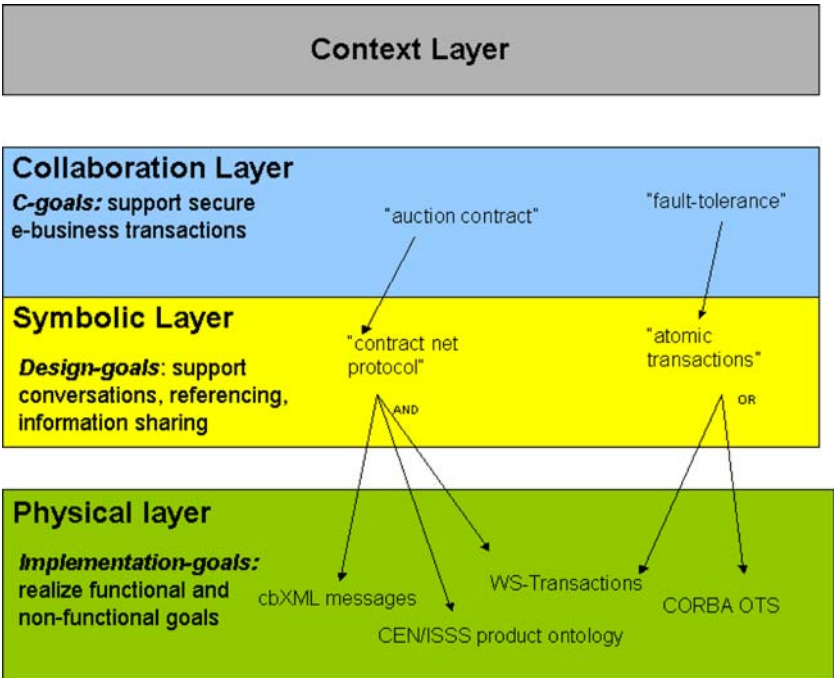

Figure 1. A two-dimensional stratified goal hierarchy.

partner. These values can be used to motivate the collaboration goals when these goals are put under discussion (cf. section 6).

The second dimension of the goal hierarchy expresses the non-functional goals such as security, fault tolerance and performance. These goals guide the various design choices that have to be made when decomposing a business transaction. Following the Model-Driven Architecture style (Kleppe, Warmer and Bast 2003), a distinction can be made between the computation-independent model (CIM), the platform-independent model (PIM) and the platform-specific model (PSM) level, that correspond directly to the layers above. CIM goals are associated to the business level and concerned with abstract goals such as a low-cost strategy and shortening delivery times. Key considerations in the second PIM layer are the coordination of the collaborative processes (centralized or decentralized coordination), and transaction management (short- or long-lived transactions, or no transactions at all). Lastly, the PSM level captures implementation choices. For example, if at the PIM level the collaborative process goal is to enable long-running transactions, at the PSM level one may choose between applying WS-Transactions, CORBA OTS or developing a proprietary transaction protocol. Note that adapting the system by replacing one component by another – e.g. a proprietary transaction protocol by WS-Transactions – is only possible when the new component fulfils the same functional goals *and* the non-functional goals (to a degree of satisfaction required at that time).

4. A Framework for Self-Adaptation

In this section, we introduce a background theory for self-adaptive systems, a taxonomy of self-adaptive systems and a framework distinguishing between first-order and second-order self-adaptation.

4.1. Adaptation in CAS

Autonomic computing has been inspired by a large body of biological theory on immune systems, pertaining to a special category of adaptive systems. Of particular relevance is the theory of Complex Adaptive Systems (CAS). A complex adaptive system is defined as a system composed of interacting agents, which respond to stimuli, exhibiting stimulus-response behavior that can be defined in terms of rules. Agents adapt by changing their rules as experience accumulates and can be aggregated into meta-agents whose behavior may be complex and emergent, i.e., not determinable by analysis of lower level agents (Holland 1995). In analogy to the human body (e.g., the human brain), autonomic system components may protect themselves to external threats and adapt themselves, without the overall system being aware of that. In case of major disruptions, autonomic components may generate algedonic signals to the system to request human intervention.

Adaptation is a central concept for CAS systems in general, and autonomic systems more in particular. In its biological context, adaptation refers to the capability of a living system to adapt to its environment. From an enterprise computing perspective, adaptive systems pertain to a special flavor of enterprise systems (or components) that may adapt themselves to changes by modifying, removing or adding new rules that govern their behavior. While improving its capacity to evolve in a changed context, the system accumulates new knowledge, learning about "good" and "bad" practices.

While evolving constantly, each CAS system occupies a special niche, which is defined by its sum of emergent behavior and characteristics. In case a CAS system ceases to exist, another system may reoccupy the niche. From an enterprise computing perspective, this may sound strange; after all, applications are being designed and deployed intentionally, to serve a specific purpose, so they should not just vanish or evolve in completely different directions. However, there is not necessarily a conflict. From the point of view of the higher-level system that deploys the component for a specific purpose, the component may seem to be completely predictable and under control. However, from an external observer perspective, the component does evolve over time (either semi-autonomously or by modifications brought about by the higher-level system, that is not visible) and it may also become obsolete and be replaced by another component.

A basic prerequisite for progressive adaptation is that individual CAS systems *learn* to avoid certain behavior in case that it does not result in the satisfaction of one of the system's (sub)goals. CAS systems are thus equipped with the capability to make inferences and predict the outcome of some actions, by comparing its

current state with that of those it has been in the past. Anticipation of events is achieved by using internal models. We may distinguish between two elementary types of internal models: tacit models and overt models. Tacit internal models are capable of producing an implicit prediction of a future state, given a current action. Tacit models are typically used by rather simple systems, without sensory capabilities. Overt models depend on more sophisticated information about the environment, acquired by sensors of the system, to explore explicit (alternative) scenarios for establishing a particular goal.

4.2. Towards an adaptation taxonomy

Adaptation may be structured along three orthogonal dimensions (see Figure 2). The first dimension captures the object of adaptation (the 'What' dimension). We distinguish between two types of artifacts that may be adapted: goals and behavior (realized functionality). This distinction can be said to be relative. However, we posit that for each system at a given point in time, the distinction can be made, and is relevant when we assess the impact of adaptation. Behavior can be further refined in tasks (e.g. a sorting task) and processes, and for processes it is useful to distinguish between internal processes not visible outside and processes in the interface that typically can be changed without affecting the core functionality.

The second perspective concentrates on the way in which we may adapt systems (the 'How' dimension). Basically, we may discern between two techniques to steer the adaptation process: reasoning and learning. Reasoning entails the ability to make predictions given a collection of facts. This typically requires an extensive causal

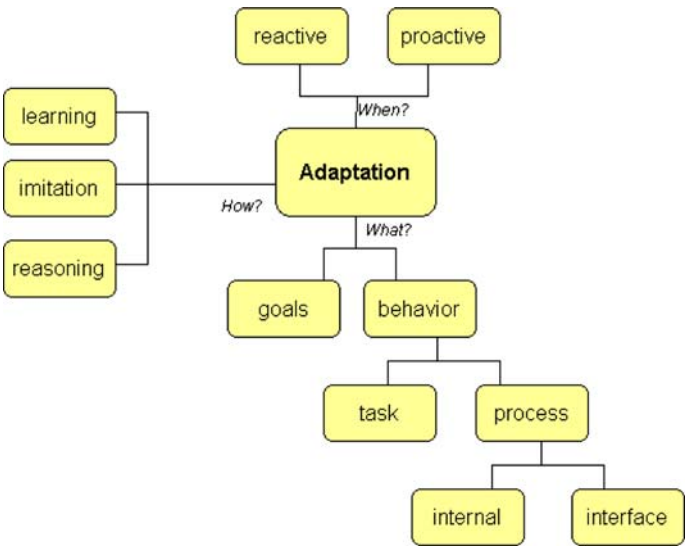

Figure 2. Adaptation taxonomy.

model, but data mining can also be used. Learning is understood here as the process of acquiring knowledge on the basis of experience, that is, trial and error. Techniques that may be adopted for learning include Bayesian rules, reinforcement learning, and evolutionary computation. Imitation is a special kind of learning based on the experience of others rather than own one's experience.

The third dimension of adaptation taxonomy emphasizes the time dimension (the 'When' dimension); adaptation may occur pro-actively or reactively. Reactive adaptation of systems is performed using predetermined responses to external signals. Proactive (pre-emptive) adaptation aims at accommodating future changes by anticipating potential scenarios, and typically relies on continuous monitoring of the environment.

4.3. Self-Adaptive systems

Adaptation of software is typically considered at three levels of granularity: (1) cross-system level, (2) system level, and (3) the component-level. Each of these levels has its specific requirements for adaptation. Regardless of these levels, adaptation may be either left to the discretion of external designers, or, performed (partially) by the system itself. The latter category of adaptation is named *self-adaptation*.

Self-adaptive systems evaluate their behavior, using reflection mechanisms, and modify *themselves* in case that their behavior does not conform to predefined goals, or, when the system can be optimized in terms of non-functional properties such as

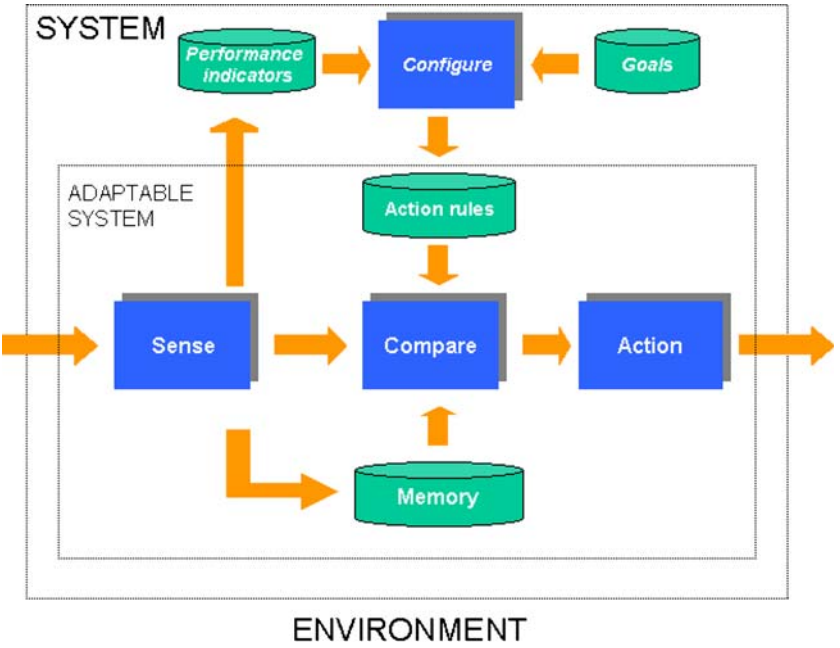

Figure 3. Self-adaptive system architecture.

performance, security and stability. We call that part of the system that is modifiable the adaptable system.

Figure 3 depicts a framework defining and relating essential concepts for self-adaptation. Systems execute *actions* that have an effect on the environment. As outlined, self-adaptive systems constantly *sense* incoming signals, and compare them with *action rules* that make up the core of the adaptable system. The *compare* component can be a complex process, during which not only the current status (from the sense data) is considered, but also a predicted status. That is why a *memory* component can be necessary. The memory component captures past states of the system, relating them to incoming and outgoing events. Having a memory component, we allow for both event-driven and state-driven comparison: in the former case, it is a particular event that may match a certain rule and trigger some action, whereas in the latter case, it is the internal model (included in the memory, and constantly adapted on the basis of incoming signals) that is compared to condition values.

So far, the system is nothing more than what is called a feedback system in standard system theory (Owens 1978) in which only the environment is adapted to. In order to introduce self-adaptation, we first distinguish between the *goals* of the system and an operational way of achieving the goals, the adaptable system. The latter can be thought of as a set of Event-Condition-Action rules, but how these action rules are implemented is not essential. The performance of the system is monitored under a *perpetual testing* strategy (Osterweil et al. 1996) in the form of *performance indicators*. In case the configuration of the adaptable system is not a one-shot event but a continuous effort to find the optimal configuration for realizing the goals, we talk about (first-order) self-adaptation. The extreme case is that after each action cycle, the system determines a new configuration of action rules based on the goals and the result (measured by the performance indicator). The determination of the configuration can be a random explorative process in the beginning, while gradually becoming more effective. This is the typical setting for reinforcement learning. A less extreme case is when the adaptable system incorporates a certain plan to achieve the goal that is not evaluated after each action cycle but only after some period or when a break-down prompts for it.

First-order self-adaptation is bound to limitations, as the goals may be infeasible or have become so. Hence alternatively, the system may adapt its goals. This requires a *second-order* adaptive system (double-loop learning, Argyris and Schön 1974). Although this kind of adaptation is often projected inside the adaptive system, thereby raising its complexity, we prefer to model it as a combination of two systems, where the higher system (manager) monitors the behavior of the lower system (agent) and subsequently may adapt the goals of the agent (Figure 4). The manager also has its own goals that it cannot change, and presumably it is itself an agent of a higher-level system, ultimately the human user. So in general, a self-adaptive system is realized not by one agent but by a hierarchically organized agent society (cf. Ciancarini, Omicini and Zambonelli 2000).

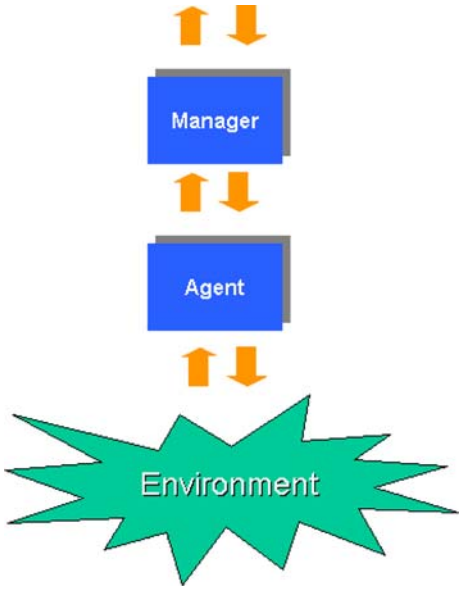

Figure 4. Second-order self-adaptive system.

5. A Framework for Co-Adaptation

Traditionally, co-adaptive systems pertain to multiple interacting constituents of a system that learn concurrently in relation to one another. Thus, co-adaptive systems denote a special type of self-adaptive systems for which the environment is not a black-box, but a gray-box including other self-adaptive systems that are embedded in the same environment. This means that self-adaptation cannot be any longer achieved in splendid isolation: adapting one system can have an impact on the other system, and vice versa. An immediate impact of adapting a system may be that the communication with the other system breaks down, completely or partially. But even if the communication does still work, the consequences of associated systems to adapt themselves independently may have unexpected negative consequences for both.

Hence there is a need for *coordinated adaptation*. We define co-adaptive systems as systems that enable coordinated adaptation through direct symbolic interaction (cf. Section 3), rather than indirectly by the effect of the actions on the environment. For interoperable systems, like the ones that support business conversations, co-adaptation is essential, as these systems depend heavily on agreed upon protocols, data standards and other shared specifications (it is exactly at the interfaces that most of the adaptations have to be made). So changes to the system, such as moving from EDIFACT to XML, from non-secure to secure connections, or from Dutch to English naming cannot be made unilaterally.

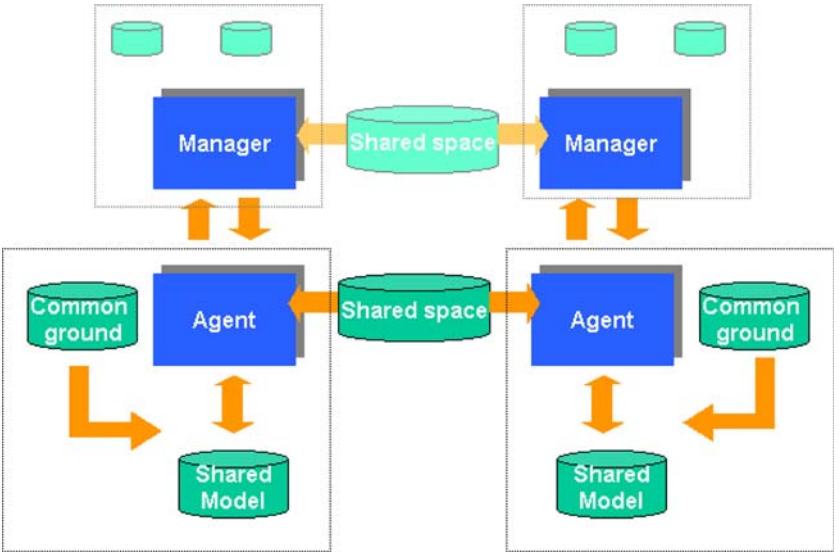

Figure 5. Co-adaptive system architecture.

For the interaction between two systems, we distinguish between the shared space, the shared model and the common ground (Figure 5). The shared space is what links the two systems together so that they can communicate at all. In a computer network, the shared space is the physical network between the two computers. The shared space may also be a physical context, as the environment where ants leave so-called pheromenes to communicate with each other. Abstracting from the devices to send and receive signals in this environment, the shared space is the backbone over which systems synchronize their shared model. It corresponds to the physical level of collaborative systems as outlined in Section 3. The shared model captures the symbolic and collaboration level and is the representation of some part of the world designated by the signals. We use the term shared model here as an abstract notion; whether there is an explicit shared model, such as the message list for two communicating computers, or not, as is probably the case with the ants, is not relevant.

The common ground is whatever the systems use in interpreting symbols. This can include a shared domain ontology, interaction protocols, and a shared set of inference rules. Again, this common ground need not be explicit to be there and to have effect, but if it is explicit and in declarative format, it is of course much easier to manipulate and adapt. Because of the common ground, the shared model will typically not contain just messages, but also several interpretations. For example, a shared view of the obligations that derive from these messages (cf. Section 2). Essentially, the common ground contains the definition of the shared space, the definition of the shared model (in terms of interpretation rules) and the mapping between the shared space and the shared model.

5.1. Levels of Co-Adaptation

We can now define the *first* level of co-adaptation to be the communication that occurs when the two systems adapt their shared model using signals transmitted in the shared space. The continuous and collaborative effort of putting signals in the shared space and taking signals out of the shared space is to be viewed as a mechanism aimed at synchronizing their shared model. A file replication mechanism in a P2P network illustrates well this level of co-adaptation (Figure 6).

Now it may be the case that inserting the signal into the shared model, interpreting it and subsequently expanding this model, raises a logical conflict (Figure 7). In our framework, these conflicts relate to the validity claims in the signal. For example, the signal may be a purchase order (ϕ) with the implicit assumption that the ordered goods are available (ψ). If the goods are not available, this claim is not valid, and an argumentation process will start to solve the issue. This *second* level of co-adaptation is worked out below. In the figure, a distinction is made between the Shared Model and the Private DB. In a co-adaptive system, the Memory component foreseen in the self-adaptive system framework can be decomposed into one or more private databases and a number of shared models (as many as there are collaborative partners) that should be mutually consistent but are not necessarily synchronized.

The *third* level of co-adaptation occurs when the common ground is adapted. A simple example is when a symbol gets a new meaning, or when the syntactic structure of the symbol changes. At this point, traditional type checking systems typically raise an error, and perhaps signal an error message to the other system.

A level3 co-adaptive system would try to adapt the common ground in such a way that the problem is solved (Figure 8). Note that this adaptation of the common ground is not needed for all breakdowns. It may also be the case that the sending system has made an error, or that an error has occurred in the shared space. In that case, the obvious solution is retransmission, which we regard as part of the level1 co-adaptation. Furthermore, it may be the case that the common ground leaves room for various interaction styles, and the only adaptation that is needed is switching from one style to another. An example is a modem that supports multiple data rates

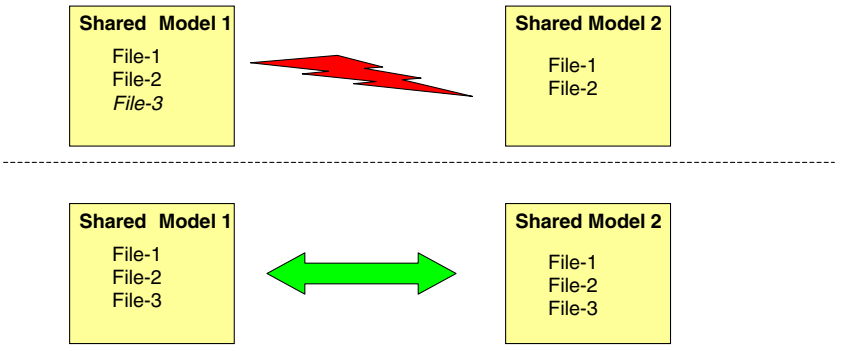

Figure 6. Level-1 co-adaptation in a P2P network.

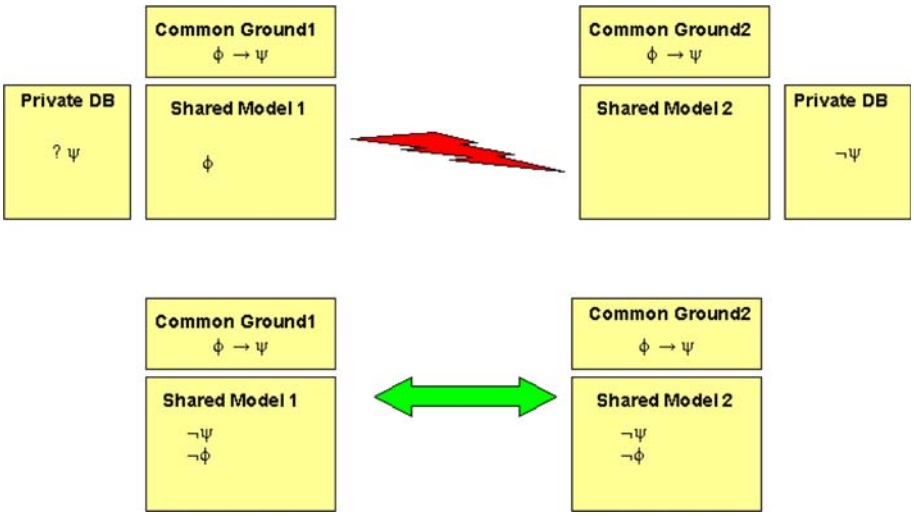

Figure 7. Level 2 co-adaptation.

and negotiates dynamically with the counter-party which data rate to use. This flexibility is built-in in the common ground and so the common ground itself need not be adapted.

Finally, it may occur that the common ground includes goals of the system that it cannot change itself. In that case, we talk about a *fourth* level of co-adaptation. It cannot be done without the active involvement of the manager of the agent. Actually, in this case a breakdown signal is reported to the manager through the agent's performance indicators. In most situations, the two agents involved will have different managers. What happens then is a co-adaptation process between the two managers,

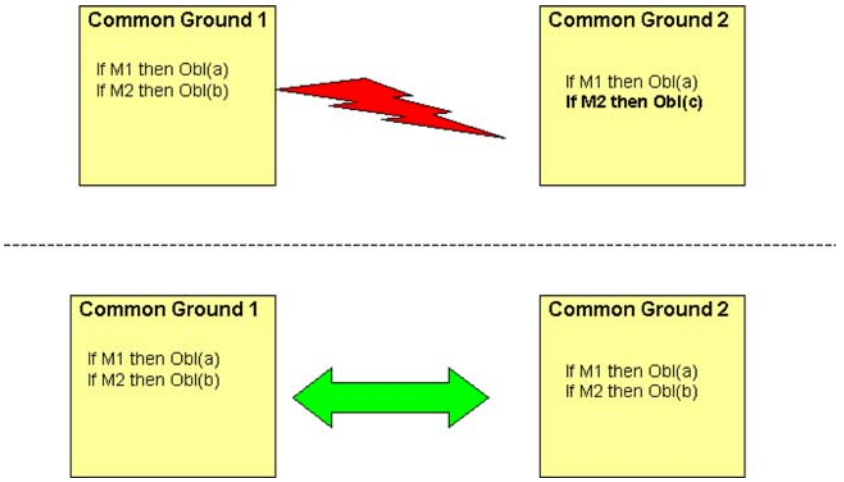

Figure 8. Level 3 adaptations.

on the basis of their common ground. The intended result of this co-adaptation process is an agreement on a new common ground, which may involve changing the goals of the agents below them. The claim that we make in this paper is that ultimately, co-adaptation should be supported on all four levels.

The basic message processing algorithms are pictured in Figure 9: one for processing operational messages and one for processing updates to the common ground. If a received message does not adhere to the agreed upon format, an error message is sent, otherwise it is accepted (level1 co-adaptation). If validity claims are not acceptable, a ''non-agree'' message is sent to the other party. This is the beginning of an argumentation process worked out in the next section (level2 co-adaptation). Otherwise, the claims are accepted and the shared models are synchronized. The second algorithm says that when an update of the common ground is received, it is either accepted (level3 co-adaptation) or forwarded to the Manager (level4 co-adaptation).

6. Argumentation

Second-level co-adaptation requires some way of synchronizing shared models for which argumentation can offer a solution. In recent years, dialogue systems for argumentation have received interest in several fields of AI, particularly in AI and law (Prakken 2000, 2001) and agent communication and negotiation (Amgoud, Belabbes and Prade 2005; Bentahar, Moulin and Chaib-dra 2003). In argumentation

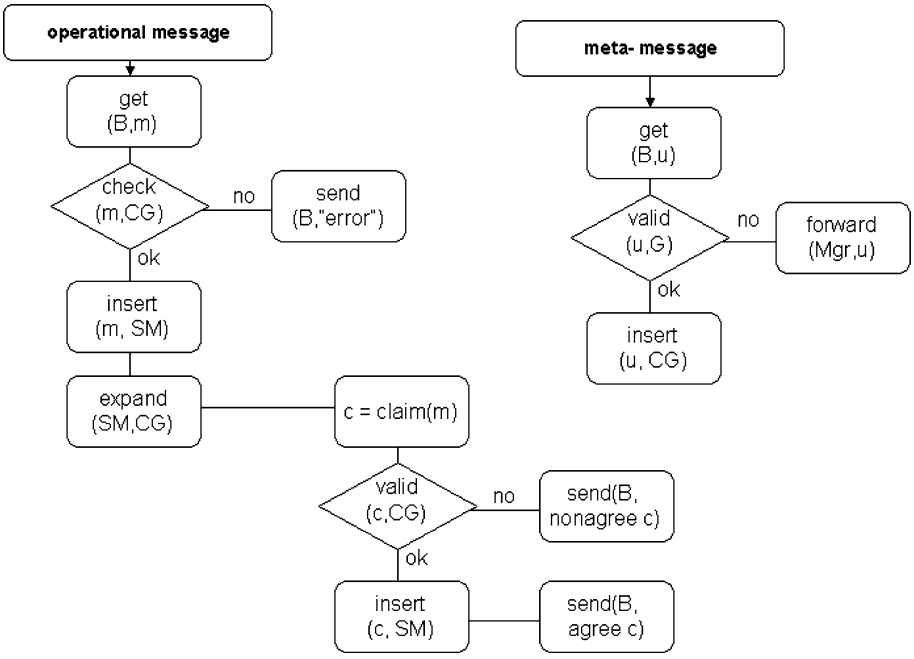

Figure 9. Basic message processing algorithms for co-adaptive systems.

theory (Toulmin 1969), formal dialogue systems have been developed for so-called "persuasion" or "critical discussion" (Mackenzie 1979; Walton and Krabbe 1995 – other types are information-seeking dialogues and inquiry dialogues). The dialogue system in this case regulates the use of speech acts for such things as making or challenging a claim, accepting, withdrawing or arguing. The proponent of a claim aims at making the opponent concede (accept) his claim; the opponent instead aims at making the proponent withdraw his claim. Such a dialogue ends when one of the players has fulfilled his aim. The following definition of a dialogue system is based on the work of Prakken referred to above and is also used in (De Moor and Weigand, fc).

6.1. A basic dialogue system

Prakken defines a dialogue system (in particular, a protocol for persuasion by dispute, PPD for short) as a tuple consisting of many elements. We have slightly adapted and simplified his system in the following definition. Our reformulation reflects the fact that in co-adaptation, the communicative action should be focused on resolving conflicts rather than winning arguments.

Definition

A *protocol for persuasion by dispute* (PPD) consists of the following elements *Players, Acts, Replies, Moves, Comms, Rules, Resolution* as defined below

Players, typically represented with the characters S and H
Acts, the set of discussion acts: claim(.φ), argue(φ, so ψ), why(φ), retract(φ), accept(φ), where φ is a wff and "ψ, so φ" is an argument.
Replies, a function that defines for each act what are the possible reply acts (see Table 1). They can be characterized as either agreeing or disagreeing.
Moves, the set of all well-formed moves. An initial move is a pair < Player, Act > , a responding move is a triple < Player, Act, Move > , where the third component indicates the move to which the current move responds.

Table 1. Argumentation acts and replies (based on Prakken 2001).

ACTS	Disagreeing replies	Agreeing replies
claim ϕ	why ϕ	accept ϕ
	argue Φ so$\neg\phi$	
why ϕ	argue Φ so ϕ	retract ϕ
accept ϕ		
retract ϕ		
argue A: Φ so ψ	argue B: ϕ' so ψ'	accept A
where A identifies	where argument B	retract C
this argument	challenges or undercuts A	where C is an argument
	why ϕ_I	challenged by A
	where $\phi_I \in \Phi$	accept ϕ_i

Comms is a function that assigns to each player at each stage of a dialogue a set of propositions to which the player is committed at that stage. At the start, these can be considered empty or equal to the *Agreed*.
Rules is a function that for any dialogue state specifies the allowed moves at that point, given the dialogue so far and the players' commitments.
Resolution, is a function that determines how the discussion result is established. One way of establishing the result is to determine who is the "winner", which is subsequently defined as the one whose argument cannot be defeated.

The *dialogical status* of the move indicates its status in the discussion. A certain claim is "in" when it has been made and not challenged. If it has been challenged, it becomes "out", until the challenge itself is effectively replied to (see the publications referred to for a formal definition). Another important element is *Comms*, the commitments of the discussion partners at that state. These commitments are to be understood here in the context of the rational discussion; they represent what the player adheres to, even if it is only for the sake of the argument (they do not correspond with responsibilities for action). During a discussion, the partners can take on various commitments, of which only a part is agreed.

The logical semantics of the possible discussion acts can be represented using *Comms*. For example, the effect of argue (φ, so ψ) is that φ and ψ are added to the commitments of the speaker. The preconditions of the discussion acts refer to the Commitments as well. One general condition is that *Comms* must be left consistent. More specific preconditions are given in the following table based on (Prakken 2001). The relationship between *Comms* and the *Agreed* (Section 2) is simply that the *Agreed* is the intersection of the two *Comms*.

Each dialogue system specifies somehow what are the allowed moves at some point (the Rules component). In the literature, no single or optimal set of rules can be found, but there are some general norms that seem to be necessary in any rational discussion: *Non-repetition*, that is, if moves m_i and m_j are both reply to M, then their content should be different; *Relevance*: a move is relevant iff it replies to a relevant target. A target is relevant iff any attacking reply to it changes the dialogical status of the initial move. Every move (except the initial move) should be relevant;

Table 2. Pre- and post-condition rules. Note that the moves can only affect the commitments of the speaker of the act, not of the hearer.

Move	Preconditions	Postcondition (effects on the speakers commitments)
claim ϕ	*Comms* $\cup \{\phi\}$ is consistent	*Comms*: = *Comms* $\cup \{\phi\}$
argue Φ so ψ		*Comms*: = *Comms* $\cup \{\psi\} \cup \Phi$
retract ϕ	$\phi \in$ *Comms* (explicitly added)	*Comms*: = *Comms*/$\{\phi\}$
accept ϕ	$\phi \notin$ *Comms* *Comms* do not justify $\neg\phi$	*Comms*: = *Comms* $\cup \{\phi\}$
why ϕ	*Comms* do not justify ϕ	(no change)

No self-contradiction: it is not allowed to concede to a proposition if the opposite is justified by the player's own commitments (it is allowed that the speaker has changed his mind, but then he should retract his earlier commitment).

6.2. Argumentation and co-adaptation

So far we have followed the main line of Prakken's work. We think that this dialogue system is in instrumental in supporting level2 co-adaptation, but in its current form it addresses formal aspects of the argumentation only. As such, it will leave many disputes unresolved. What typically happens is that a dialogue starts, and the parties exchange some arguments until they arrive at one or more propositions p that are not derivable anymore (basic beliefs). Suppose the parties disagree on the value of p. The dialogue system does not provide a solution (unless the Resolution has some arbitrary decision rule, e.g. that in such a case, the first claimant always wins, but this is not desirable).

In order to increase the chances of successful resolution, we propose to add *data authority rules* to the common ground. A data authority rule states which agent is an authorative source for a certain class of propositions. For example, it can be stated that each agent is authorative with respect to data concerning itself or the company it represents. For certain type of data, third parties may be declared to be authorative. In this way, many conflicts can be resolved. There are a few special cases to consider.

- *incompleteness*. If there is no data authority rule for some proposition p, then we adopt the following rule; if the parties make contradictory claims, the issue remains unresolved. If one party makes a claim and the other does not want to claim the opposite (because he simply does not know – for him, the truth value is unknown), that means that the other should concede.
- *inconsistency*. If according to the data authority rules both parties are authorative, then the same rule is applied: the issue remains unresolved. Note that this means that overlapping authority rules are as ''bad'' as incomplete rules, and hence it does not make much sense to have them. However, the criteria used in the data authority rules can be diverse, and so it may be hard to exclude overlapping a priori.

Furthermore, we note that our co-adaptation framework allows for escalation. When a certain issue remains unresolved between two agents, this is considered as a breakdown that triggers the Managers to start a dialogue. Using both data authority rules and escalation, the number of unresolved issues can be reduced drastically. The remaining ones are brought to the attention of the human user.

Let us illustrate by a simple example how the argumentation framework supports co-adaptation. Consider the level2 co-adaptation in Figure 7. The Buyer claims that the order must be processed (ϕ), assuming that the goods are available (ψ). Both parties agree that the claim is only valid when the goods are available ($\phi \rightarrow \psi$). When Seller receives and interprets the claim, he will notice the logical conflict

($\psi \neg \psi$). According to Table 1, he can reply with a why or an argue move. Since he has an argument, he replies with the move, "argue Φ so$\neg\phi$", where Φ is:$\{\phi \rightarrow \psi, \neg\psi\}$. As Buyer agrees on the first part of .Φ and has no way of attacking the second part (he does not know the status of ψ, and realizes that Seller is authorative on this proposition), he can only reply by an accept move, in which he accepts the argue move and hence also adds$\neg\psi$ to his commitments, that is, his part of the Shared Model in terms of section 5.

7. Conclusion

In this paper, we have made a first step towards a generic co-adaptive framework for business collaboration that is not only able to support operational e-commerce transactions but also able to adapt the collaboration definition itself. In particular, this paper distinguishes between self-adaptation and co-adaptation, and introduces a stratified framework for each. A limitation of the paper is that the aspects of adaptability, self-adaptation and co-adaptation are not integrated yet.

Most current work on self-adaptation focuses on individual systems only, although configurations of self-adaptive components are considered as well. However, in a shared environment, self-adaptive systems better coordinate their adaptation efforts. This is the more true when we realize that systems usually do not exist for their own sake, but to serve another system. So we suggest that self-adaptation should not be studied only at the micro level of an individual system, but also at the macro level of interacting systems. Interesting questions here are: What are emergent properties? What can make the macro system unstable? What infrastructure is desirable (e.g. matching functions, contracting mechanisms) to support co-adaptation and its initialization? Answers to these questions may be found in the area of Multi-Agent Systems, where these issues have been dealt with for some time.

The advantage of using a co-adaptive system architecture is not only the increase in autonomy of the system, and an accompanying reduction of maintenance costs. It can also make the specification more transparent, as all the self-adaptation and co-adaptation is already built-in, and need not be specified anymore by the developer. The developer can concentrate on the goals that he wants to set, including the performance indicators.

The co-adaptation framework presented here constitutes an initial research product in nature. In the future, we wish to develop formal underpinnings for the various types of adaptation. Also, we plan to extend the framework with other generic mechanisms. It is fair to say that a main emphasis of the research efforts in the field of Information Systems for the past few decades was to find generic *representations*, such as conceptual data models, generic ontologies and general-purpose communication languages. This approach has been proven to be quite successful, but we believe it is not sufficient, at least not for realizing the autonomous computing vision. Therefore, we envision a new challenge for the coming decade: to complement generic representations with generic *mechanisms* of which self-adaptation, co-adaptation and imitation are good examples to start with.

References

Amgoud, L., S. Belabbes, H. Prade. (2005). "Towards a Formal Framework for the Search of a Consensus between Autonomous Agents," In: *Proc. of the Fourth Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, The Netherlands, July 25–29, 2005, AAMAS '05. ACM Press, New York.

Argyris, C. and D. Schön (1974). *Theory in Practice: Increasing Professional Effectiveness*. San Francisco: Jossey-Bass.

Bentahar, J., B. Moulin, and B. Chaib-dra. (2003). "Commitment and Argument Network: A New Formalism for Agent Communication," in: AAMAS Workshop on Agent Communication Languages and Conversation Policies, Melbourne, 2003.

Chaibdra, B. and F. Dignum (2002). "Trends in Agent Communication Language,"*Computational Intelligence* 18(2), 89–101.

Cheong, C. and M. Winikoff. (2005). "Hermes: a Methodology for Goal Oriented Agent Interactions," in: Proceedings of the Fourth Int. Joint Conf. on Autonomous Agents and Multiagent Systems, The Netherlands, July 25–29, 2005, AAMAS '05. ACM Press, New York.

Ciancarini, P., A. Omicini, and F. Zambonelli. (2000). "Multiagent Systems Engineering: the Coordination Viewpoint," in: Intelligents Agents VI (ATAL99), LNAI. Springer-Verlag, Berlin.

Clark, H. (1996). Using Language. Cambridge University Press.

Dardenne, A., A. van Lamsweerde and S. Fickas (1993). "Goal-Directed Requirements Acquisition,"*Science of Computer Programming* 20(1–2), 3–50.

Dietz, J. L. G. (2005). *Enterprise Ontology – Theory and Methodology*. Berlin: Springer-Verlag.

Ganek, A. G. and T. A. Corbi (2003). "The Dawning of the Autonomic Computer Area,"*IBM Systems Journal* 42(1), 5–18.

Garlan, D. and B. Schmerl. (2002). "Model-based Adaptation for Self-Healing Systems," in: 1st Workshop on Self-Healing Systems ACM Press, New York.

Habermas, J. (1984). The Theory of Communicative Action, I. Beacon Press.

Holland, J. H. (1995). *Hidden Order: How Adaptation builds Complexity*. CambridgeMA: Perseus Books.

Jones, A. J. and X. Parent. (2003). "Conventional Signalling Acts and Conversation," in: AAMAS Workshop on Agent Communication Languages and Conversation Policies, Melbourne, 2003.

Kephart, O. and D. M. Chess (2003). "The Vision of Autonomic Computing,"*IEEE Computer* 36(1), 41–50.

Kimbrough, S. O. and S. A. Moore (1997). "On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech act Theory and Expressive Felicity,"*ACM Transactions on Information Systems* 15(4), 321–367.

Kleppe, A. G., J. Warmer, and W. Bast. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc.

van Lamsweerde, A. (2001). "Goal-Oriented Requirements Engineering: A Guided Tour," in: Proc. RE'01, Toronto, August 2001, pp. 249–263.

Linington, P. F., Z. Milosevic, J. Cole, S. Gibson, S. Kulkami and S. Neal (2004). "A Unified Behavioural Model and a Contract Language for Extended Enterprise,"*Data and Knowledge Engineering* 51(1), 5–30.

Mackenzie, J. D. (1979). "Question-Begging in Non-Cumulative Systems,"*Journal of Philosophical Logic* 8, 117–133.

Mylopoulos, J., L. Chung and E. Yu (1999). "From Object-oriented to Goal-oriented Requirements Analysis,"*Communications of the ACM* 42(1), 31–37.

de Moor, A. and H. Weigand. (fc). Formalizing the Evolution of Virtual Communities, Information Systems. (accepted for publication).

Norman, D. A. (1990). *The Design of Everyday Things*. New York: Doubleday.

Osterweil, L. J., L. A. Clarke, D. J. Richardson, and M. Young. (1996). "Perpetual Testing," in: Proceedings of the Ninth International Software Quality Week.

Owens, D. H. (1978). *Feedback and Multivariate Systems*. Stevenage: Peter Peregrinus Ltd.

Pfleeger, S. L. and J. M. Atlee. (2006). Software Engineering – Theory and Practice. 3rd ed., Pearson Int.

Prakken, H. (2000). "On Dialogue Systems with Speech Acts, Arguments and Counterarguments," In: *Proc. of the 7th European Workshop on Logic for Artificial Intelligence (JELIA*'2000), LNAI 1919, Springer-Verlag, pp. 224–238.

Prakken, H. (2001). "Modeling Reasoning about Evidence in Legal Procedure (2001)," in: Proc. ICAIL-2001, St. Louis, Missouri.

Salehie, M. and L. Tahvildari. (2005). ''Autonomic Computing: Emerging Trends and Open Problems,'' in: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software (St. Louis, Missouri, May 21–21, 2005). DEAS '05. ACM Press, New York.

Singh, M. P. (2000). ''Social Semantics for Agent Communication Languages''. in F. Dignum, & M. Greaves, (Eds) *Issues in Agent Communication* (pp. 31–45). Berlin: Springer-Verlag.

Smith, H. and P. Fingar. (2003). *Business Process Management: The Third Wave*. Meghan-Kiffer Press.

Stamper, R. (2000). ''New Directions for Systems Analysis and Design''. in J. Filipe (ed.) *Enterprise Information Systems* (pp. 14–39). London: Kluwer Academic Publ..

Toulmin, S. E. (1969). *The Uses of Argument*. Cambridge University Press.

Valetto, G. and G. Kaiser (2002). *A Case Study in Software Adaptation*. *Proc. 1st Workshop on Self-Healing Systems*. New York: ACM Press.

Walton, D. N. and E. Krabbe (1995). *Commitment in Dialogue*. *Basic Concepts of Interpersonal Reasoning*. Albany NY: State Univ of New York Press.

Weigand, H., E. Verharen, and F. Dignum. (1997). ''Integrated Semantics for Information and Communication Systems,'' In: Meersman, R. and L. Mark. (eds.) *Proc. of IFIP DS-6 ''Database Application Semantics*,'' Stone-Mountain, Georgia.

Weigand, H. and W. J. van den Heuvel (1999). ''Meta-Patterns for Electronic Commerce Transactions based on FLBC,''*International Journal on Electronic Commerce* 3(2), 45–66.